DISCRETE MATHEMATICS

W W L CHEN

Chapter 8

TURING MACHINES

8.1. Introduction

A Turing machine is a machine that exists only in the mind. It can be thought of as an extended and modified version of a finite state machine.

Imagine a machine which moves along an infinitely long tape which has been divided into boxes, each marked with an element of a finite alphabet A. Also, at any stage, the machine can be in any of a finite number of states, while at the same time positioned at one of the boxes. Now, depending on the element of A in the box and depending on the state of the machine, we have the following:

- The machine either leaves the element of A in the box unchanged, or replaces it with another element of A.
- The machine then moves to one of the two neighbouring boxes.
- The machine either remains in the same state, or changes to one of the other states.

The behaviour of the machine is usually described by a table.

Example 8.1.1. Suppose that $A = \{0, 1, 2, 3\}$, and that we have the situation below:

$$\begin{array}{c} 5 \\ \downarrow \\ \ldots\ 003211\ \ldots \end{array}$$

Here the diagram denotes that the Turing machine is positioned at the box with entry 3 and that it is in state 5. We can now study the table which describes the bahaviour of the machine. In the table below, the column on the left represents the possible states of the Turing machine, while the row at the top represents the alphabet (*i.e.* the entries in the boxes):

	0	1	2	3
⋮				
5				$1L2$
⋮				

The information "$1L2$" can be interpreted in the following way: The machine replaces the digit 3 by the digit 1 in the box, moves one position to the left, and changes to state 2. We now have the following situation:

$$\begin{array}{c} 2 \\ \downarrow \\ \ldots\ 001211\ \ldots \end{array}$$

We may now ask when the machine will halt. The simple answer is that the machine may not halt at all. However, we can halt it by sending it to a non-existent state.

EXAMPLE 8.1.2. Suppose that $A = \{0, 1\}$, and that we have the situation below:

$$\begin{array}{c} 2 \\ \downarrow \\ \ldots\ 001011\ \ldots \end{array}$$

Suppose further that the behaviour of the Turing machine is described by the table below:

	0	1
0	$1R0$	$1R4$
1	$0R3$	$0R0$
2	$1R3$	$1R1$
3	$0L1$	$1L2$

Then the following happens successively:

$$\begin{array}{c} 3 \\ \downarrow \\ \ldots\ 011011\ \ldots \end{array}$$

$$\begin{array}{c} 2 \\ \downarrow \\ \ldots\ 011011\ \ldots \end{array}$$

$$\begin{array}{c} 1 \\ \downarrow \\ \ldots\ 011011\ \ldots \end{array}$$

$$\begin{array}{c} 0 \\ \downarrow \\ \ldots\ 010011\ \ldots \end{array}$$

$$\begin{array}{c} 0 \\ \downarrow \\ \ldots\ 010111\ \ldots \end{array}$$

$$\begin{array}{c} 4 \\ \downarrow \\ \ldots\ 010111\ \ldots \end{array}$$

The Turing machine now halts. On the other hand, suppose that we start from the situation below:

$$\begin{array}{c} 3 \\ \downarrow \\ \ldots\ 001011\ \ldots \end{array}$$

If the behaviour of the machine is described by the same table above, then the following happens successively:

$$\begin{array}{c} 1 \\ \downarrow \\ \ldots\ 001011\ \ldots \end{array}$$

$$\begin{array}{c} 3 \\ \downarrow \\ \ldots\ 001011\ \ldots \end{array}$$

The behaviour now repeats, and the machine does not halt.

8.2. Design of Turing Machines

We shall adopt the following convention:

(1) The states of a k-state Turing machine will be represented by the numbers $0, 1, \ldots, k-1$, while the non-existent state, denoted by k, will denote the halting state.

(2) We shall use the alphabet $A = \{0, 1\}$, and represent natural numbers in the unary system. In other words, a tape consisting entirely of 0's will represent the integer 0, while a string of n 1's will represent the integer n. Numbers are also interspaced by single 0's. For example, the string

$$\ldots 011111101111111111010011111101011 10\ldots$$

will represent the sequence $\ldots, 6, 10, 1, 0, 6, 1, 3, \ldots$. Note that there are two successive zeros in the string above. They denote the number 0 in between.

(3) We also adopt the convention that if the tape does not consist entirely of 0's, then the Turing machine starts at the left-most 1.

(4) The Turing machine starts at state 0.

EXAMPLE 8.2.1. We wish to design a Turing machine to compute the function $f(n) = 1$ for every $n \in \mathbb{N} \cup \{0\}$. We therefore wish to turn the situation

$$\begin{array}{c} \downarrow \\ 0\underbrace{1\ \ldots\ 1}_{n}0 \end{array}$$

to the situation

$$\begin{array}{c}\downarrow\\ 010\end{array}$$

(here the vertical arrow denotes the position of the Turing machine). This can be achieved if the Turing machine is designed to change 1's to 0's as it moves towards the right; stop when it encounters the digit 0, replacing it with a single digit 1; and then correctly positioning itself and move on to the halting state. This can be achieved by the table below:

	0	1
0	$1L1$	$0R0$
1	$0R2$	

Note that the blank denotes a combination that is never reached.

Example 8.2.2. We wish to design a Turing machine to compute the function $f(n) = n + 1$ for every $n \in \mathbb{N} \cup \{0\}$. We therefore wish to turn the situation

$$\begin{array}{l}\quad\downarrow\\ 0\underbrace{1\ \ \ldots\ \ 1}_{n}0\end{array}$$

to the situation

$$\begin{array}{l}\quad\downarrow\\ 0\underbrace{1\ \ \ldots\ \ 1}_{n+1}0\end{array}$$

(*i.e.* we wish to add an extra 1). This can be achieved if the Turing machine is designed to move towards the right, keeping 1's as 1's; stop when it encounters the digit 0, replacing it with a digit 1; move towards the left to correctly position itself at the first 1; and then move on to the halting state. This can be achieved by the table below:

	0	1
0	$1L1$	$1R0$
1	$0R2$	$1L1$

Example 8.2.3. We wish to design a Turing machine to compute the function $f(n, m) = n + m$ for every $n, m \in \mathbb{N}$. We therefore wish to turn the situation

$$\begin{array}{l}\quad\downarrow\\ 0\underbrace{1\ \ \ldots\ \ 1}_{n}0\underbrace{1\ \ \ldots\ \ 1}_{m}0\end{array}$$

to the situation

$$\begin{array}{l}\quad\downarrow\\ 0\underbrace{1\ \ \ldots\ \ 1}_{n}\underbrace{1\ \ \ldots\ \ 1}_{m}0\end{array}$$

(*i.e.* we wish to remove the 0 in the middle). This can be achieved if the Turing machine is designed to move towards the right, keeping 1's as 1's; stop when it encounters the digit 0, replacing it with a digit

1; move towards the left and replace the left-most 1 by 0; move one position to the right; and then move on to the halting state. This can be achieved by the table below:

	0	1
0	$1L1$	$1R0$
1	$0R2$	$1L1$
2		$0R3$

The same result can be achieved if the Turing machine is designed to change the first 1 to 0; move to the right, keeping 1's as 1's; change the first 0 it encounters to 1; move to the left to position itself at the left-most 1 remaining; and then move to the halting state. This can be achieved by the table below:

	0	1
0		$0R1$
1	$1L2$	$1R1$
2	$0R3$	$1L2$

Example 8.2.4. We wish to design a Turing machine to compute the function $f(n) = (n, n)$ for every $n \in \mathbb{N}$. We therefore wish to turn the situation

$$0\underbrace{\overset{\downarrow}{1}\ \ldots\ 1}_{n}0$$

to the situation

$$0\underbrace{\overset{\downarrow}{1}\ \ldots\ 1}_{n}0\underbrace{1\ \ldots\ 1}_{n}0$$

(*i.e.* put down an extra string of 1's of equal length). This turns out to be rather complicated. However, the ideas are rather simple. We shall turn the situation

$$0\underbrace{\overset{\downarrow}{1}\ \ldots\ 1}_{n}0$$

to the situation

$$0\underbrace{1\ \ldots\ 1}_{n}\overset{\downarrow}{0}\underbrace{1\ \ldots\ 1}_{n}0$$

by splitting this up further by the inductive step of turning the situation

$$0\underbrace{1\ \ldots\ 1}_{r}\underbrace{\overset{\downarrow}{1}\ \ldots\ 1}_{n-r}0\underbrace{1\ \ldots\ 1}_{r}0$$

to the situation

$$0\underbrace{1\ \ldots\ 1}_{r+1}\underbrace{\overset{\downarrow}{1}\ \ldots\ 1}_{n-r-1}0\underbrace{1\ \ldots\ 1}_{r+1}0$$

(*i.e.* adding a 1 on the right-hand block and moving the machine one position to the right). This can be achieved by the table below:

	0	1
0		$0R1$
1	$0R2$	$1R1$
2	$1L3$	$1R2$
3	$0L4$	$1L3$
4	$1R0$	$1L4$

Note that this is a "loop". Note also that we have turned the first 1 to a 0 to use it as a "marker", and then turn it back to 1 later on. It remains to turn the situation

$$\begin{array}{c} \qquad\qquad\quad\downarrow \\ 0\underbrace{1\ \ \ldots\ \ 1}_{n}0\underbrace{1\ \ \ldots\ \ 1}_{n}0 \end{array}$$

to the situation

$$\begin{array}{c} \downarrow\qquad\qquad\qquad\quad \\ 0\underbrace{1\ \ \ldots\ \ 1}_{n}0\underbrace{1\ \ \ldots\ \ 1}_{n}0 \end{array}$$

(*i.e.* move the Turing machine to its final position). This can be achieved by the table below (note that we begin this last part at state 0):

	0	1
0	$0L5$	
5	$0R6$	$1L5$

It now follows that the whole procedure can be achieved by the table below:

	0	1
0	$0L5$	$0R1$
1	$0R2$	$1R1$
2	$1L3$	$1R2$
3	$0L4$	$1L3$
4	$1R0$	$1L4$
5	$0R6$	$1L5$

8.3. Combining Turing Machines

Example 8.2.4 can be thought of as a situation of having combined two Turing machines using the same alphabet, while carefully labelling the different states. This can be made more precise. Suppose that two Turing machines M_1 and M_2 have k_1 and k_2 states respectively. We can denote the states of M_1 in the usual way by $0, 1, \ldots, k_1 - 1$, with k_1 representing the halting state. We can also change the notation in M_2 by denoting the states by $k_1, k_1 + 1, \ldots, k_1 + k_2 - 1$, with k_1 and $k_1 + k_2$ representing respectively the starting state and halting state of M_2. Then the composition M_1M_2 of the two Turing machines (first M_1 then M_2) can be described by placing the table for M_2 below the table for M_1.

EXAMPLE 8.3.1. We wish to design a Turing machine to compute the function $f(n) = 2n$ for every $n \in \mathbb{N}$. We can first use the Turing machine in Example 8.2.4 to obtain the pair (n, n) and then use the Turing machine in Example 8.2.3 to add n and n. We therefore conclude that either table below is appropriate:

	0	1
0	$0L5$	$0R1$
1	$0R2$	$1R1$
2	$1L3$	$1R2$
3	$0L4$	$1L3$
4	$1R0$	$1L4$
5	$0R6$	$1L5$
6	$1L7$	$1R6$
7	$0R8$	$1L7$
8		$0R9$

	0	1
0	$0L5$	$0R1$
1	$0R2$	$1R1$
2	$1L3$	$1R2$
3	$0L4$	$1L3$
4	$1R0$	$1L4$
5	$0R6$	$1L5$
6		$0R7$
7	$1L8$	$1R7$
8	$0R9$	$1L8$

8.4. The Busy Beaver Problem

There are clearly Turing machines that do not halt if we start from a blank tape. For example, those Turing machines that do not contain a halting state will go on for ever. On the other hand, there are also clearly Turing machines which halt if we start from a blank tape. For example, any Turing machine where the next state in every instruction sends the Turing machine to a halting state will halt after one step.

For any positive integer $n \in \mathbb{N}$, consider the collection $\mathcal{T}_n$ of all binary Turing machines of n states. It is easy to see that there are $2n$ instructions of the type aDb, where $a \in \{0, 1\}$, $D \in \{L, R\}$ and $b \in \{0, 1, \ldots, n\}$, with the convention that state n represents the halting state. The number of choices for any particular instruction is therefore $4(n+1)$. It follows that there are at most $(4(n+1))^{2n}$ different Turing machines of n states, so that $\mathcal{T}_n$ is a finite collection.

Consider now the subcollection $\mathcal{H}_n$ of all Turing machines in $\mathcal{T}_n$ which will halt if we start from a blank tape. Clearly $\mathcal{H}_n$ is a finite collection. It is therefore theoretically possible to count exactly how many steps each Turing machine in $\mathcal{H}_n$ takes before it halts, having started from a blank tape. We now let $\beta(n)$ denote the largest count among all Turing machines in $\mathcal{H}_n$. To be more precise, for any Turing machine M which halts when starting from a blank tape, let $B(M)$ denote the number of steps M takes before it halts. Then

$$\beta(n) = \max_{M \in \mathcal{H}_n} B(M).$$

The function $\beta : \mathbb{N} \to \mathbb{N}$ is known as the busy beaver function. Our problem is to determine whether there is a computing programme which will give the value $\beta(n)$ for every $n \in \mathbb{N}$.

It turns out that it is logically impossible to have such a computing programme. In other words, the function $\beta : \mathbb{N} \to \mathbb{N}$ is non-computable. Note that we are not saying that we cannot compute $\beta(n)$ for any particular $n \in \mathbb{N}$. What we are saying is that there cannot exist one computing programme that will give $\beta(n)$ for every $n \in \mathbb{N}$.

We shall only attempt to sketch the proof here.

The first step is to observe that the function $\beta : \mathbb{N} \to \mathbb{N}$ is strictly increasing. To see that, we shall show that $\beta(n+1) > \beta(n)$ for every $n \in \mathbb{N}$. Suppose that $M \in \mathcal{H}_n$ satisfies $B(M) = \beta(n)$. We shall use this Turing machine M to construct a Turing machine $M' \in \mathcal{H}_{n+1}$. If state n is the halting state of M, then all we need to do to construct M' is to add some extra instruction like

n	$1L(n+1)$	$1L(n+1)$

to the Turing machine M. If $n+1$ denotes the halting state of M', then clearly M' halts, and $B(M') = \beta(n)+1$. On the other hand, we must have $B(M') \leq \beta(n+1)$. The inequality $\beta(n+1) > \beta(n)$ follows immediately.

The second step is to observe that any computing programme on any computer can be described in terms of a Turing machine, so it suffices to show that no Turing machine can exist to calculate $\beta(n)$ for every $n \in \mathbb{N}$.

The third step is to create a collection of Turing machines as follows:

(1) Following Example 8.2.2, we have a Turing machine M_1 to compute the function $f(n) = n+1$ for every $n \in \mathbb{N} \cup \{0\}$, where M_1 has 2 states.
(2) Following Example 8.3.1, we have a Turing machine M_2 to compute the function $f(n) = 2n$ for every $n \in \mathbb{N}$, where M_2 has 9 states.
(3) Suppose on the contrary that a Turing machine M exists to compute the function $\beta(n)$ for every $n \in \mathbb{N}$, and that M has k states.
(4) We then create a Turing machine $S_i = M_1 M_2^i M$; in other words, the Turing machine S_i is obtained by starting with M_1, followed by i copies of M_2 and then by M. The Turing machine S_i, when started with a blank tape, will clearly halt with the value $\beta(2^i)$; in other words, with $\beta(2^i)$ successive 1's on the tape. This will take at least $\beta(2^i)$ steps to achieve.

However, what we have constructed is a Turing machine S_i with $2+9i+k$ states and which will halt only after at least $\beta(2^i)$ steps. It follows that

$$\beta(2^i) \leq \beta(2+9i+k). \tag{1}$$

However, the expression $2+9i+k$ is linear in i, so that $2^i > 2+9i+k$ for all sufficiently large i. It follows that (1) contradicts our earlier observation that the function $\beta : \mathbb{N} \to \mathbb{N}$ is strictly increasing. The contradiction arises from the assumption that a Turing machine of the type M exists.

8.5. The Halting Problem

Related to the busy beaver problem above is the halting problem. Is there a computing programme which can determine, given any Turing machine and any input, whether the Turing machine will halt?

Again, it turns out that it is logically impossible to have such a computing programme. As before, it suffices to show that no Turing machine can exist to undertake this task. We shall confine our discussion to sketching a proof.

Suppose on the contrary that such a Turing machine S exists. Then given any $n \in \mathbb{N}$, we can use S to examine the finitely many Turing machines in $\mathcal{T}_n$ and determine all those which will halt when started with a blank tape. These will then form the subcollection $\mathcal{H}_n$. We can then simulate the running of each of these Turing machines in $\mathcal{H}_n$ to determine the value of $\beta(n)$. In short, the existence of the Turing machine S will imply the existence of a computing programme to calculate $\beta(n)$ for every $n \in \mathbb{N}$. It therefore follows from our observation in the last section that S cannot possibly exist.

Problems for Chapter 8

1. Design a Turing machine to compute the function $f(n) = n - 3$ for every $n \in \{4, 5, 6, \ldots\}$.

2. Design a Turing machine to compute the function $f(n, m) = (n + m, 1)$ for every $n, m \in \mathbb{N} \cup \{0\}$.

3. Design a Turing machine to compute the function $f(n) = 3n$ for every $n \in \mathbb{N}$.

4. Design a Turing machine to compute the function $f(n_1, \ldots, n_k) = n_1 + \ldots + n_k + k$ for every $k, n_1, \ldots, n_k \in \mathbb{N}$.

5. Let $k \in \mathbb{N}$ be fixed. Design a Turing machine to compute the function $f(n_1, \ldots, n_k) = n_1 + \ldots + n_k + k$ for every $n_1, \ldots, n_k \in \mathbb{N} \cup \{0\}$.